Analytica — An Experiment in Combining Theorem Proving and Symbolic Computation

Edmund Clarke Xudong Zhao

October 1992
CMU-CS-92-147

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597 and in part by National Science Foundation under Contract Number CCR-9005992.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. government.

Keywords: Theorem Prover, Symbolic computation, Mathemmatica, Analytica

Abstract

Analytica is an automatic theorem prover for theorems in elementary analysis. The prover is written in Mathematica language and runs in the Mathematica environment. The goal of the project is to use a powerful symbolic computation system to prove theorems that are beyond the scope of previous automatic theorem provers. The theorem prover is also able to guarantee the correctness of certain steps that are made by the symbolic computation system and therefore prevent common errors like division by a symbolic expression that could be zero.

In this paper we describe the structure of Analytica and explain the main techniques that it uses to construct proofs. Analytica has been able to prove several non-trivial examples including the basic properties of the stereographic projection and a series of three lemmas that lead to a proof of Weierstrass's example of a continuous nowhere differentiable function. Each of the lemmas in the latter example is proved completely automatically.

1 Introduction

Current automatic theorem provers, particularly those based on some variant of resolution, have concentrated on obtaining ever higher inference rates by using clever programming techniques, parallelism, etc. We believe that this approach is unlikely to lead to a useful system for actually doing mathematics. The main problem is the large amount of domain knowledge that is required for even the simplest proofs. In this paper, we describe an alternative approach that involves combining an automatic theorem prover with a symbolic computation system. The theorem prover, which we call *Analytica*, is able to exploit the mathematical knowledge that is built into this symbolic computation system. In addition, it can guarantee the correctness of certain steps that are made by the symbolic computation system and, therefore, prevent common errors like division by an expression that may be zero.

Analytica is written in the Mathematica programming language and runs in the interactive environment provided by this system [19]. Since we wanted to generate proofs that were similar to proofs constructed by humans, we have used a variant of the sequent calculus [9, 10] in the inference phase of our theorem prover. However, quantifiers are handled by skolemization instead of explicit quantifier introduction and elimination rules. Although inequalities play a key role in all of analysis, Mathematica is only able to handle very simple numeric inequalities. We have developed a technique that is complete for linear inequalities and is able to handle a large class of non-linear inequalities as well. This technique is more closely related to the BOUNDER system developed at MIT [16] than to the traditional SUP-INF method of Bledsoe [5]. Another important component of Analytica deals with expressions involving summation and product operators. A large number of rules are devoted to the basic properties of these operators. We have also integrated Gosper's algorithm for hypergeometric sums with the other summation rules, since it can be used to find closed form representations for a wide class of summations that occur in practice.

There has been relatively little research on theorem proving in analysis. Bledsoe's work in this area [3, 4] is certainly the best known. Analytica has been heavily influenced by his research. More recently, Farmer, Guttman, and Thayer at Mitre Corporation [8] have developed an interactive theorem prover for analysis proofs that is based on a simple type theory. Neither of these uses a symbolic computation system for manipulating mathematical formulas, however. Suppes and Takahashi [17] have combined a resolution

theorem prover with the *Reduce* system, but their prover is only able to check very small steps and does not appear to have been able to handle very complicated proofs. London and Musser [14] have also experimented with the use of Reduce for program verification.

Our paper is organized as follows: In Section 2, we give two simple examples that illustrate the power of our theorem prover and show how it uses various symbolic computation techniques provided by Mathematica. Section 3 contains an overview of the structure of Analytica and the major techniques that it uses in constructing proofs. Sections 4 and 5 describe several of the most important techniques in greater detail. Section 4 deals with summation and includes a short description of how we have integrated Gosper's algorithm into the prover. Section 5 discusses how Analytica treats inequalities. The paper concludes in Section 6 with a discussion of some extensions that we hope to add to Analytica in the near future.

2 Simple examples proved by Analytica

In each example, the input for the prover is given first. The theorem and its proof are printed by the theorem prover. Mathematica automatically generates Latex commands to typeset formulas involving algebraic expressions.
1. The sum of two roots of a quadratic equation.

```
Prove[imp[and[a!=0, x!=y, a x^2 + b x + c == 0, a y^2 + b y + c == 0],
        x + y == -b/a]]
```

Theorem :

$$\left(a \neq 0 \wedge x \neq y \wedge ax^2 + bx + c = 0 \wedge ay^2 + by + c = 0 \Rightarrow x + y = -\frac{b}{a}\right)$$

Proof :

$$a \neq 0 \wedge x \neq y \wedge c + bx + ax^2 = 0 \wedge c + by + ay^2 = 0 \Longrightarrow x + y = -\frac{b}{a}$$

reduces to

$$c + bx + ax^2 = 0 \wedge c + by + ay^2 = 0 \Longrightarrow x = y \vee a = 0 \vee x + y = -\frac{b}{a}$$

rewrite as

$$c + bx + ax^2 = 0 \wedge c + by + ay^2 = 0 \Longrightarrow x - y = 0 \vee a = 0 \vee \frac{b + ax + ay}{a} = 0$$

reduces to

$$c + bx + ax^2 = 0 \wedge c + by + ay^2 = 0 \Longrightarrow x - y = 0 \vee a = 0 \vee b + a(x + y) = 0$$

solve linear equation

$$c = -(x(b+ax)) \wedge c = -(y(b+ay)) \Longrightarrow x - y = 0 \vee a = 0 \vee b + a(x+y) = 0$$

substitute using equation

$$-(x(b+ax)) = -(y(b+ay)) \Longrightarrow x - y = 0 \vee a = 0 \vee b + a(x+y) = 0$$

reduces to

$$x(b+ax) = y(b+ay) \Longrightarrow x - y = 0 \vee a = 0 \vee b + a(x+y) = 0$$

rewrite as

$$(x-y)(b+ax+ay) = 0 \Longrightarrow x - y = 0 \vee a = 0 \vee b + ax + ay = 0$$

reduces to

$$x - y = 0 \vee b + a(x+y) = 0 \Longrightarrow x - y = 0 \vee a = 0 \vee b + a(x+y) = 0$$

simplify formula using local context

$$True$$

□

2. Closed form for a summation.

```
Prove[imp[and[integer[n], 0<=n, m!=1], sum[2^k/(1+m^(2^k)), {k, 0, n}] ==
                                    1/(m-1) + 2^(n+1)/(1-m^(2^(n+1)))]];
```

Theorem :

$$(integer(n) \wedge 0 \leq n \wedge m \neq 1 \Rightarrow \sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}} = \frac{1}{m-1} + \frac{2^{n+1}}{1-m^{2^{n+1}}})$$

Proof :

$$integer(n) \wedge 0 \leq n \wedge m \neq 1 \Longrightarrow \sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}} = \frac{1}{-1+m} + \frac{2 \cdot 2^n}{1-m^{2 \cdot 2^n}}$$

reduces to

$$integer(n) \wedge 0 \leq n \Longrightarrow m = 1 \vee \sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}} = \frac{1}{-1+m} + \frac{2 \cdot 2^n}{1-m^{2 \cdot 2^n}}$$

prove

$$\sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}} = \frac{1}{-1+m} + \frac{2 \cdot 2^n}{1-m^{2 \cdot 2^n}}$$

use induction on n

base case with n = 0

$$m = 1 \vee \frac{1}{1+m} = \frac{1}{-1+m} + \frac{2}{1-m^2}$$

reduces to

$$True$$

induction step

$$integer(n) \wedge 0 \leq n \wedge \sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}} = \frac{1}{-1+m} + \frac{2 \cdot 2^n}{1-m^{2 \cdot 2^n}} \Longrightarrow$$

$$m = 1 \vee \sum_{k=0}^{1+n} \frac{2^k}{1+m^{2^k}} = \frac{1}{-1+m} + \frac{4 \cdot 2^n}{1-m^{4 \cdot 2^n}}$$

calculate summations

$$integer(n) \wedge 0 \leq n \wedge \sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}} = \frac{1}{-1+m} + \frac{2 \cdot 2^n}{1-m^{2 \cdot 2^n}} \Longrightarrow$$

$$m = 1 \vee \frac{2 \cdot 2^n}{1+m^{2 \cdot 2^n}} + \left(\sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}} \right) = \frac{1}{-1+m} + \frac{4 \cdot 2^n}{1-m^{4 \cdot 2^n}}$$

substitute using equation

$$integer(n) \wedge 0 \leq n \wedge \sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}} = \frac{1}{-1+m} + \frac{2 \cdot 2^n}{1-m^{2 \cdot 2^n}} \Longrightarrow$$

$$m = 1 \vee \frac{2 \cdot 2^n}{1+m^{2 \cdot 2^n}} + \frac{1}{-1+m} + \frac{2 \cdot 2^n}{1-m^{2 \cdot 2^n}} = \frac{1}{-1+m} + \frac{4 \cdot 2^n}{1-m^{4 \cdot 2^n}}$$

reduces to

$$True$$

□

3 An overview of Analytica

Analytica consists of four different phases: skolemization, simplification, inference, and rewriting. When a new formula is submitted to Analytica for proof, it is first skolemized to a quantifier free form. Then it is simplified using a collection of algebraic and logical reduction rules. If the formula reduces to true, the current branch of the inference tree terminates with success. If not, the theorem prover checks to see if the formula matches the conclusion of some inference rule. If a match is found, Analytica will try to establish the hypothesis of the rule. If the hypothesis consists of a single formula, then it will try to prove that formula. If the hypothesis consists of a series of formulas, then Analytica will attempt to prove each of the formulas in sequential order. If no inference rule is applicable, then various rewrite rules are used attempting to convert the formula to another

equivalent form. If the rewriting phase is unsuccessful, the search terminates in failure; otherwise the simplification, inference and rewriting phases will repeat with the new formula. Backtracking will cause the entire inference tree to be searched before the proof of the original goal formula terminates with failure.

3.1 Skolemization phase

In Analytica (as in Bledsoe's *UT Prover* [3]), we use skolemization to deal with the quantifiers that occur in the formula to be proved. Initially, quantified variables are standardized so that each has a unique name. We define the *position* of a quantifier within a formula as *positive* if it is in the scope of even number of negations, and *negative* otherwise. *Skolemization* consists of the following procedure: Replace $(\exists x.\Psi(x))$ at positive positions or $(\forall x.\Psi(x))$ at negative positions by $(\Psi(f(y_1, y_2, ..., y_n)))$ where $x, y_1, y_2, ..., y_n$ are all the free variables in $\Psi(x)$ and f is a new function symbol, called a *skolem function.* The original formula is satisfiable if and only if its skolemized form is satisfiable. Thus, X is valid if and only if X' is valid where $\neg X'$ is the skolemized form of $\neg X$ [9]. We call $\neg skolemize(\neg f)$ the *negatively skolemized form* of f . A formula is valid if and only its negatively skolemized form is valid. When a negatively skolemized formula is put in prefix form, all quantifiers are existential. These quantifiers are implicitly represented by marking the corresponding quantified variables. The marked variables introduced by this process are called *skolem variables.* The resulting formula will be quantifier-free. For example, the skolemized form of the formula

$$(\exists x.\forall y.P(x,y)) \rightarrow (\exists u.\forall v.Q(u,v))$$

is given by

$$P(x, y_0(x)) \rightarrow Q(u_0(), v),$$

while its negatively skolemized form is

$$P(x_0(), y) \rightarrow Q(u, v_0(u)).$$

where x, y, u and v are skolem variables, and u_0, v_0, x_0, y_0 are skolem functions. Although formulas are represented internally in skolemized form without quantifiers, quantifiers are added when a formula is displayed so that proofs will be easier to read.

3.2 Simplification phase

Simplification is the key phase of Analytica. A formula is simplified with respect to its *proof context*. Intuitively, the proof context consists of the formulas that may be assumed true when the formula is encountered in the proof. The formula that results from simplifying f under context C is denoted by $\mathit{simplify}(f, C)$. In order for the simplification procedure to be sound, $\mathit{simplify}(f, C)$ must always satisfy the the following condition

$$C \models \mathit{simplify}(f, C) \leftrightarrow f.$$

The initial context C_0 in each simplification phase is a conjunction of all of the *given* properties of the variables and constants in the theorem. The initial formula in each simplification phase is the current goal of the theorem prover. In the first simplification phase it is the result of the skolemization phase. In each subsequent simplification phase it is the result of the previous rewriting phase. The simplification procedure for composite formulas is given by the following rules:

1. $\mathit{simplify}(f) = \mathit{simplify}(f, C_0)$
2. $\mathit{simplify}(f_1 \wedge f_2, C) = f_1' \wedge \mathit{simplify}(f_2, C \wedge f_1')$
   where $f_1' = \mathit{simplify}(f_1, C \wedge f_2)$
3. $\mathit{simplify}(f_1 \vee f_2, C) = f_1' \vee \mathit{simplify}(f_2, C \wedge \neg f_1')$
   where $f_1' = \mathit{simplify}(f_1, C \wedge \neg f_2)$
4. $\mathit{simplify}(f_1 \rightarrow f_2, C) = f_1' \rightarrow \mathit{simplify}(f_2, C \wedge f_1')$
   where $f_1' = \mathit{simplify}(f_1, C \wedge \neg f_2)$
5. $\mathit{simplify}(\neg f, C) = \neg \mathit{simplify}(f, C)$

The soundness of these rules can be easily established by structural induction. For example, if the soundness condition holds for f_1 and f_2, it will also hold for $f_1 \wedge f_2$, etc.

A large number of rules are provided for simplifying atomic formulas (i.e., equations and inequalities) using context information. Some examples of rules for simplifying inequalities are given in Section 5. In addition to the equation and inequality rules, special simplification rules are included to handle functions that are frequently used, such as Abs, Min, Max, Sum, Product, Limit, etc. The simplification of summations and products is discussed in detail in Section 4.

The following example illustrates how the context information is used to simplifying formulas:

Theorem :

$$(0 < a < b \Rightarrow b^3 - a^3 > (b-a)^3)$$

Proof :

$$0 < a < b \Longrightarrow -a^3 + b^3 > (-a+b)^3$$

reduces to

$$0 < a \wedge a - b < 0 \Longrightarrow 3a\,(a-b)\,b < 0$$

reduces to

$$0 < a \wedge a - b < 0 \implies (0 < b \wedge -a + b < 0 \vee a - b < 0 \wedge b < 0) \wedge a < 0 \vee 0 < a \wedge (0 < b \wedge a - b < 0 \vee -a + b < 0 \wedge b < 0)$$

simplify formula using context information

$$0 < a \wedge a - b < 0 \Longrightarrow 0 < b$$

replace expression with its lower or upper bounds

$$0 < a \wedge a - b < 0 \Longrightarrow 0 \leq a$$

reduces to

True

□

3.3 Inference phase

The inference phase is based on the *sequent calculus* [10]. We selected this approach because we wanted our proofs to be readable. Suppose that f is the formula that we want to prove. In this phase we attempt to find an instantiation for the skolem variables that makes f a valid ground formula. In order to accomplish this, f is decomposed into a set of *sequents* using rules of the sequent calculus. Each sequent has the form $\Gamma \vdash \Delta$, where Γ and Δ are initially sets of subformulas of f. The formula f will be proved, if substitution can be found that makes all of the sequents valid. A sequent $\Gamma \vdash \Delta$ is valid if it is impossible to make all of the elements of Γ true and all of the elements of Δ false.

In Analytica, the funcition *FindSubstitution*(f) is used to determine the appropriate substitution for f. If f is not provable, *FindSubstitution*(f) will return *Fail*. *FindSubstitution* has rules corresponding to each of the rules of the sequent calculus except those concerning quantifiers. The two rules for implication are given as examples:

1. Implification on the left:

   $\mathit{FindSubstitution}(\Gamma, A \rightarrow B, \Delta \vdash \Lambda) = \sigma_1\sigma_2$ where
   $\sigma_1 = \mathit{FindSubstitution}(\Gamma, \Delta \vdash A, \Lambda)$, and
   $\sigma_2 = \mathit{FindSubstitution}(\Gamma\sigma_1, B\sigma_1, \Delta\sigma_1 \vdash \Lambda\sigma_1)$.

2. Implication on the right:

   $\mathit{FindSubstitution}(\Gamma \vdash \Delta, A \rightarrow B, \Lambda) = \mathit{FindSubstitution}(\Gamma, A \vdash \Delta, B, \Lambda)$

Rules are also needed for atomic formulas. The three below are typical.

1. Equation: $\mathit{FindSubstitution}(\Gamma \vdash \Delta, a = b, \Lambda) = \sigma$ where $a\sigma = b\sigma$.

2. Inequality: $\mathit{FindSubstitution}(\Gamma, a < b, \Delta \vdash \Lambda) = \sigma$ where $a\sigma = b\sigma$.

3. Matching: $\mathit{FindSubstitution}(\Gamma, A, \Delta \vdash \Lambda, B, \Theta) = \sigma$ where $A\sigma = B\sigma$.

Backtracking is often necessary in the inference phase when there are multiple subgoals, because a substitution that makes one subgoal valid may not make another subgoal valid. When this happens it is necessary to find another substitution for the first subgoal. In order to restart the inference phase at the correct point, a stack is added to the procedure described above. When a rule is applied that may generate several subgoals, one subgoal is selected as the current goal and the others are saved on the stack. If some substitution σ makes the current subgoal valid, then σ is applied to the other subgoals on the stack and Analytica attempts to prove them. If the other subgoals are not valid under σ, then Analytica returns to the previous goal and tries to find another substitution that makes it valid.

Special tactics are included in the inference phase for handling inequalities and constructing inductive proofs. The tactic that is used for inequalities is described in detail in Section 5 and will not be discussed further here. The induction tactic enables Analytica to select a suitable induction scheme for the formula to be proved and attempts to establish the basis and induction steps. A typical induction scheme is

$$f(n_0) \wedge \forall n(n \geq n_0 \wedge f(n) \rightarrow f(n+1)) \Rightarrow \forall n(n \geq n_0 \rightarrow f(n))$$

In this case, we need only to identify the induction variable n and determine the base value for n. In order to find a suitable induction variable for formula f, we list all variables that appear in f and select those that have type integer. To reduce the search space, we would like to make sure that our choice of the induction variable is a good one. The choice is good if the

induction hypothesis is useful for proving the induction conclusion. This will be more likely if the terms that appear in the induction conclusion appear either in the induction hypothesis or in the current context. Hence, we arrive at the following heuristic for selecting the induction variable: Use n as the induction variable to prove $f(n)$ provided that $f(n+1)$ only contains terms that already appear in $f(n)$ or in the current context. Once the induction variable n has been selected, a base value for that variable must be found in order to start the induction. In Analytica, a suitable base value may be determined by calculating the set of lower bounds of n as described in Section 5 and choosing the simplest element of this set. If the basi case fails for this value, Analytica will choose another base value and try again until the basis is proven or no other choice is available. In the former case, the induction step is tried; otherwise the induction scheme fails and Analytica will try other techniques like those in the rewriting phase. This strategy is used in the constructing the induction proof for the second example in Section 2.

3.4 Rewrite phase

Five rewriting tactics are used in Analytica:

1. When the left hand side of an equation in the hypothesis appears in the sequent, it is replaced by the right hand side of the equation. For example,

$$\sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}} = \frac{1}{-1+m} + \frac{2\cdot 2^n}{1-m^{2\cdot 2^n}} \Longrightarrow$$
$$\frac{2\cdot 2^n}{1+m^{2\cdot 2^n}} + \left(\sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}}\right) = \frac{1}{-1+m} + \frac{4\cdot 2^n}{1-m^{4\cdot 2^n}}$$

   `substitute using equation`

$$\sum_{k=0}^{n} \frac{2^k}{1+m^{2^k}} = \frac{1}{-1+m} + \frac{2\cdot 2^n}{1-m^{2\cdot 2^n}} \Longrightarrow$$
$$\frac{2\cdot 2^n}{1+m^{2\cdot 2^n}} + \frac{1}{-1+m} + \frac{2\cdot 2^n}{1-m^{2\cdot 2^n}} = \frac{1}{-1+m} + \frac{4\cdot 2^n}{1-m^{4\cdot 2^n}}$$

2. Rewrite a trigonometric expression to an equivalent form.

Given that a is an odd integer, k, m, n are integers, $m \leq n$,

$$-\cos(\pi a^n x) + (-1)^k \cos(\pi a^{-m+n}(a^m x - k)) = 0$$

rewrite trigonometric expressions

$$True$$

3. Move all terms in equations or inequalities to left hand side and factor the expression.

$$\frac{(-1+x_3)^2\,(-1+{y_2}^2+{y_3}^2)}{(-1+y_3)^2} = -1+{x_3}^2+\frac{(-1+x_3)^2{y_2}^2}{(-1+y_3)^2}$$

rewrite as

$$\frac{2(-1+x_3)(x_3-y_3)}{-1+y_3} = 0$$

4. Solve linear equations.

$$c+bx+ax^2 = 0 \wedge c+by+ay^2 = 0 \Longrightarrow x-y = 0 \vee b+a(x+y) = 0$$

solve linear equation

$$c = -(x(b+ax)) \wedge c = -(y(b+ay)) \Longrightarrow x-y = 0 \vee b+a(x+y) = 0$$

5. Replace a user defined function by its definition. In the example below the user defined function S is expanded.

$$0 < \pi a^m b^m + (1-ab)\,\mathrm{Abs}(S(m))$$

expand definition

$$0 < \pi a^m b^m + (1-ab)\,\mathrm{Abs}\Big(\sum_{n=0}^{-1+m} \frac{b^n\,(-\cos(\pi a^n x)+\cos(\pi a^n\,(x+h)))}{h}\Big)$$

4 Summation

Summations play an important role in symbolic computation. Nevertheless, Mathematica's ability to handle summations is very limited. A summation with range from n_1 to n_2, where n_1 and n_2 are integers and $n_1 \leq n_2$, is explicitly expanded into a sum with $n_2 - n_1 + 1$ terms. However, a

summation with a symbolic range will not be simplified. Consequently, we have introduced a large number of special rules for dealing with summations. Although most of the rules are based on simple identities, Analytica is able to handle a large range of summations in example proofs. Analogous rules for products are also included in Analytica. A few of the rules for summation are listed below. The rules are partitioned into three sets.

1. The first set of rules reduces the number of summations occurring in the expression to be simplified.

   $\sum_{n=n_1}^{n_2} c = c(n_2 - n_1 + 1)$ where c is a constant

   $\sum_{n=n_1}^{n_2} f_1(n) + \sum_{n=n_1}^{n_2} f_2(n) = \sum_{n=n_1}^{n_2} (f_1(n) + f_2(n))$

   $\sum_{n=n_1}^{n_2} f(n) + \sum_{n=n_2+1}^{n_3} f(n) = \sum_{n=n_1}^{n_3} f(n)$

   $\sum_{n=n_1}^{n_3} f(n) - \sum_{n=n_1}^{n_2} f(n) = \sum_{n=n_2+1}^{n_3} f(n)$

   $\sum_{n=n_1}^{n_3} f(n) - \sum_{n=n_2}^{n_3} f(n) = \sum_{n=n_1+1}^{n_2} f(n)$

2. The second set does not change the number of summations, but simplifies summands.

   $\sum_{n=n_1}^{n_2} cf(n) = c \sum_{n=n_1}^{n_2} f(n)$ where c is a constant

   $\sum_{k=n_1}^{n_2} f(k+1) = \sum_{k=n_1+1}^{n_2+1} f(k)$

   $\sum_{k=n_1}^{n_2} f(k-1) = \sum_{k=n_1-1}^{n_2-1} f(k)$

3. The third set does not change the number of summations or the summands, but simplifies the ranges.

   $\sum_{n=n_1}^{n_2} f(n) = -\sum_{n=n_2+1}^{n_1-1} f(n)$ if $n_1 > n_2$

   $\sum_{n=n_1}^{n_2+N} f(n) = (\sum_{n=n_1}^{n_2} f(n)) + f(n_2+1) + ... + f(n_2+N)$

   $\sum_{n=n_1}^{n_2-N} f(n) = (\sum_{n=n_1}^{n_2} f(n)) - f(n_2) - ... - f(n_2-N+1)$

   where N is positive integer

4.1 A summation example

The following example comes from a lemma used in the proof of the existence of a continuous, nowhere differentiable function given by Weierstrass. [18]

$$\sum_{n=0}^{\infty} b^n \cos(\pi a^n x) - (-1)^{\alpha} \left(\sum_{n=m}^{\infty} b^n \left(1 + \cos(\pi a^{-m+n} \xi(m))\right) \right) - \sum_{n=0}^{\infty} b^n \cos(\pi a^{-m+n}(1+\alpha))$$

$$+\sum_{n=0}^{-1+m} b^n\left(-\cos(\pi a^n x)+\cos(\pi a^{-m+n}(1+\alpha))\right)=0$$

simplify summations

$$-\left((-1)^\alpha\left(\sum_{n=m}^{\infty} b^n\left(1+\cos(\pi a^{-m+n}\xi(m))\right)\right)\right)+\sum_{n=0}^{\infty} b^n\left(\cos(\pi a^n x)-\cos(\pi a^{-m+n}(1+\alpha))\right)$$
$$+\sum_{n=0}^{-1+m} b^n\left(-\cos(\pi a^n x)+\cos(\pi a^{-m+n}(1+\alpha))\right)=0$$

simplify summations

$$-\left((-1)^\alpha\left(\sum_{n=m}^{\infty} b^n\left(1+\cos(\pi a^{-m+n}\xi(m))\right)\right)\right)+\sum_{n=m}^{\infty} b^n\left(\cos(\pi a^n x)-\cos(\pi a^{-m+n}(1+\alpha))\right)=0$$

simplify summations

$$\sum_{n=m}^{\infty}(-(-1)^\alpha b^n\left(1+\cos(\pi a^{-m+n}\xi(m))\right)+b^n\left(\cos(\pi a^n x)-\cos(\pi a^{-m+n}(1+\alpha))\right))=0$$

reduces to

$$\sum_{n=m}^{\infty}(b^n\left(-\cos(\pi a^n x)+(-1)^\alpha\left(1+\cos(\pi a^{-m+n}\xi(m))\right)+\cos(\pi a^{-m+n}(1+\alpha))\right))=0$$

This can be simplified to True by trigonometric rules.

4.2 Gosper's Algorithm

In many examples, it would be helpful if we could obtain a closed form representation for some summation. ***Gosper's algorithm*** is able to compute such a representation for a large class of summations. Consequently, we have also integrated this method into our theorem prover. A function g is said to be a *hypergeometric function* if $g(n+1)/g(n)$ is a rational function of n. Gosper's algorithm is able to find a closed form for the series $\sum_{k=1}^{n} a_k$ when there is a hypergeometric function that satisfies $g(n)=\sum_{k=1}^{n} a_k+g(0)$ [13]. The following example illustrates how Gosper's algorithm is used in Analytica:

Theorem :

$$(|x|>1 \Rightarrow \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k^2+(2x^2+1)k+x^2(x^2+1)}\right)<\frac{1}{2})$$

Proof :

$$|x|>1 \Longrightarrow \lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k^2+x^2(1+x^2)+k(1+2x^2)}\right)<\frac{1}{2}$$

reduces to

$$1-|x|<0 \Longrightarrow -\frac{1}{2}+\lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{(k+x^2)(1+k+x^2)}\right)<0$$

calculate summation with Gosper's Algorithm

$$1-|x|<0 \Longrightarrow -\frac{1}{2}+\lim_{n\to\infty}\left(\frac{1}{2+x^2}+\frac{1}{(1+x^2)(2+x^2)}-\frac{1}{1+n+x^2}\right)<0$$

simplify limits

$$1-|x|<0 \Longrightarrow -\frac{1}{2}+\frac{1}{2+x^2}+\frac{1}{(1+x^2)(2+x^2)}<0$$

reduces to

$$1-|x|<0 \Longrightarrow \frac{1-x^2}{2+2x^2}<0$$

reduces to

$$1-|x|<0 \Longrightarrow 1-x^2<0$$

replace expression with its lower or upper bounds

True

□

5 Inequalities

Inequalities play a key role in all areas of analysis. Since Mathematica does not provide any facility for handling inequalities, we have built several techniques into Analytica for reasoning about them.

5.1 Simplification of inequalities

There are many rules that simplify atomic formulas involving inequalities. However, we only include four examples.

1. $\mathit{simplify}(0 \le a^n, C) = \mathit{True}$ if $\mathit{simplify}(0 < a, C) = \mathit{True}$
2. $\mathit{simplify}(0 < a^n, C) = \mathit{True}$ if $\mathit{simplify}(0 < a, C) = \mathit{True}$
3. $\mathit{simplify}(a^n \le 0, C) = \mathit{False}$ if $\mathit{simplify}(0 < a, C) = \mathit{True}$
4. $\mathit{simplify}(a^n < 0, C) = \mathit{False}$ if $\mathit{simplify}(0 < a, C) = \mathit{True}$

There are also rules that use upper and lower bound information to simplify inequalities. If a has a negative upper bound, then $a < 0$ is true, while $a > 0$ and $a = 0$ are both false. The function *Lower(Upper)* gives a set of lower(upper) bounds for its argument and will be discussed in Section 5.3. The set of lower(upper) bounds is calculated in the current context.

1. $\mathit{simplify}(f_1 \leq f_2, C) = \mathit{False}$ if $\exists x[x \in \mathit{Lower}(f_1 - f_2, C) \wedge x > 0]$.
2. $\mathit{simplify}(f_1 \leq f_2, C) = \mathit{True}$ if $\exists x[x \in \mathit{Lower}(f_2 - f_1, C) \wedge x \geq 0]$.
3. $\mathit{simplify}(f_1 < f_2, C) = \mathit{True}$ if $\exists x[x \in \mathit{Lower}(f_2 - f_1, C) \wedge x > 0]$.
4. $\mathit{simplify}(f_1 < f_2, C) = \mathit{False}$ if $\exists x[x \in \mathit{Lower}(f_1 - f_2, C) \wedge x \geq 0]$.

5.2 Proof strategy for Inequalities

Although many inequality formulas can be handled in the simplification phase, some valid inequality formulas cannot be reduced to true in this phase. For example, $(a \leq 0 \wedge b \leq a) \rightarrow b \leq 0$ cannot be proved by the technique used in simplification phase alone. Other more powerful techniques for deciding satisfiability of inequality formulas must be used in addition. If the inequality $a \leq b$ is not directly provable using the techniques in the simplification phase, then Analytica will try to find a term c, such that $a \leq c$ and $c \leq b$ are both provable in the current context. In order to find such a term c, we compute a set of upper bounds for a and a set of lower bounds for b by using information provided by the current context. The sets computed are denoted by $\mathit{Upper}(a)$ and $\mathit{Lower}(b)$, respectively. A term x will be in $\mathit{Upper}(a)$ only if $a <= x$ is true in the current context. Likewise, x will be in $\mathit{Lower}(b)$ only if $x <= b$ is true in the current context. To prove $a \leq b$, it is sufficient to prove that there is some $c \in \mathit{Upper}(a)$ such that $c \leq b$ is true or that there is some $c \in \mathit{Lower}(b)$ such that $a \leq c$ is true.

In order to deal with strict inequalities, we introduce a new symbol S such that both $S_L(a) \leq b$ and $a \leq S_U(b)$ are equivalent to $a < b$. Hence, $S_U(x) \in \mathit{Upper}(a)$ only if $a < x$ is true in the current context, and $S_L(x) \in \mathit{Lower}(a)$ only if $x < a$ is true in the current context. $S_U(a) + b = S_U(a + b)$ because $c \leq S_U(a + b)$ iff $c < a + b$ iff $c - b < a$ iff $c - b \leq S_U(a)$ iff $c \leq S_U(a) + b$. Similarly, $S_L(a) + b = S_L(a + b)$, $-S_L(a) = S_U(-a)$ and $-S_U(a) = S_L(-a)$, etc. This convention permits both strict inequalities and nonstrict inequalities to be handled by the same method.

It is possible to show that the technique is complete for linear inequalities, and it can also be used to prove many of the nonlinear inequalities that arise

in practice. The technique is not guaranteed to be complete for nonlinear inequalities, however.

5.3 Calculating upper and lower bounds for expressions

There are three main ways of obtaining upper and lower bounds for expressions.

1. Obtain bounds from context information:

   Upper and lower bounds for an expression are calculated in the current context. For example, when proving $(a \leq b) \vee c$, the upper bounds of a and the lower bounds of b are calculated under the context of $\neg c$. In general, If $a \leq b$ is a conjunct of the current context, we have

   $$a \in \mathit{Lower}(b), \;\; b \in \mathit{Upper}(a),$$

   and if $a < b$ is a conjunct of the current context, we have

   $$S_L(a) \in \mathit{Lower}(a), \;\; S_U(b) \in \mathit{Upper}(a).$$

2. Obtain bounds from the monotonicity of some function:

   If f is a monotonically increasing function, and a' is an upper(lower) bound of a, $f(a')$ is an upper(lower) bound of $f(a)$; if f is a monotonically decreasing function and a' is an upper(lower) bound of a, $f(a')$ is a lower(upper) bound of $f(a)$. For example:

   $$\{cx | x \in \mathit{Upper}(a)\} \subseteq \mathit{Lower}(ca), \text{ if } \; c \leq 0$$

3. Use some known bound on the value of a function:

   If f is bounded, i.e. for all x, $f(x) \leq M$, or $f(x) \geq M'$, M is an upper bound for $f(x)$ and M' a lower bound for $f(x)$. For example:

   $$x + \frac{1}{2} \in \mathit{Upper}(\mathit{round}(x))$$

   $$x - \frac{1}{2} \in \mathit{Lower}(\mathit{round}(x))$$

5.4 An example to illustrate inequality proofs

The following example also comes from the proof of the Weierstrass theorem mentioned earlier. Assume that $b > 0$,

$$2b^m - \frac{3\left(\sum_{n=m}^{\infty} b^n \left(1 + \cos(\pi a^{-m+n}(a^m - round(a^m)))\right)\right)}{1 - (a^m - round(a^m))} \leq 0$$

replace expression with its lower or upper bounds

$$2b^m - \frac{3b^m\left(1 + \cos(\pi(a^m - round(a^m)))\right)}{1 - (a^m - round(a^m))} \leq 0$$

reduces to

$$2 - \frac{3\left(1 + \cos(\pi(a^m - round(a^m)))\right)}{1 - (a^m - round(a^m))} \leq 0$$

replace expression with its lower or upper bounds

$$-2\cos(\pi(a^m - round(a^m))) \leq 0$$

reduces to

$$0 \leq \cos(\pi(a^m - round(a^m)))$$

The last inequality will be reduced to True in the rewriting phase by using the tactic for trigonometric identities.

6 Conclusion

In a related project that we plan to describe in a forthcoming paper, we have managed to prove all of the theorems and examples in Chapter 2 of Ramanujan's Collected Works[2] completely automatically. The techniques that we use are similar to those described in this paper. We believe that the examples that we have been able to prove provide convincing justification for combining powerful symbolic computation techniques with theorem provers.

Nevertheless, there are many ways to improve Analytica. One direction is to add powerful algorithmic techniques for simplifying particular classes of formulas (like extensions of Gosper's algorithm for summations). The difficulty with adding such techniques is that a proof obtained in this manner may be virtually impossible for a human to follow.

Another direction is to strengthen the ability of Analytica to do inductive proofs. The technique that Analytica currently uses for generating induction schemes is quite simple. More research is needed on the generation of complex induction schemes and the identification of sufficiently general hypotheses for inductive proofs. There has been a fair amount of research on this problem [6, 7], but more work should be done in the context of inductive proofs in analysis.

Most proofs in modern analysis are based on set theory and many use topological concepts. Clearly, the extension of Analytica to handle such proofs is critical. Although theorem proving in set theory has been an important problem for a long time, there is no generally accepted technique for constructing such proofs. The most successful work on set theory so far is probably that of Quaife [15]. His work, however, uses a theorem prover based on hyper-resolution and may not produce proofs that are very readable.

Better methods for managing hypotheses and previously proved lemmas and theorems are also needed. Techniques developed for proof checking systems like LCF [12] and HOL [11] may be adequate in the short run, but some type of higher-order unification or matching will probably be necessary in the majority of cases. In general, deciding when to use an hypothesis or previous result is a very difficult problem. Every student of elementary calculus learns the mean value theorem by heart, but giving a good set of rules for determining when to apply this theorem in order to obtain a simpler bound on some complicated expression is not easy.

Certainly, some type of higher order logic would be more appropriate for analysis than the first order logic we currently use. The ability to state higher-order lemmas would be an additional advantage of basing the prover on a higher order logic and might help solve the problem described in the last paragraph. We intend to experiment with combining ideas from this paper with Andrews' theorem prover for higher order logic [1] in the near future.

Perhaps, the most serious problem in building a theorem prover like Analytica is the soundness of the underlying symbolic computation system. Mathematica (as well as Macsyma, Reduce, and Maple) has some rules that lead to correct results in most cases but do not lead to correct results all the time. We believe the solution to the soundness problem is to develop the theorem prover and the symbolic computation system together so that each simplification step can be rigorously justified.

References

[1] P.B.Andrews, *On Connections and Higher-Order Logic*, Journal of Automated Reasoning 5, 1989, pp257-291.

[2] B.C.Berndt, ***Ramanujan's Notebooks, Part I***, Springer-Verlag, 1985, pp 25-43.

[3] W.W. Bledsoe, *The UT Natural Deduction Prover*, Technique Report ATP-17B, Mathematical Dept., University of Texas at Austin, 1983.

[4] W.W.Bledsoe, *Some Automatic Proofs in Analysis*, Contemporary Mathematics, Vol.29, 1984.

[5] W.W.Bledsoe, P.Bruell, and R.Shostak, *A Prover for General Inequalities*, Technique Report ATP-40A, Mathematical Dept., University of Texas at Austin, 1979.

[6] R.S.Boyer and J.S.Moore, *A Computational Logic*, Academic Press, 1979.

[7] A.Bundy, F.van Harmelen, J.Hesketh, and A.Smaill, *Experiments with Proof Plans for Induction*, Technique Report, Department of Artificial Intelligence, University of Edinburgh, 1988.

[8] W.M.Farmer, J.D.Guttman and F.J.Thayer, *IMPS: AN Interactive Mathematical Proof System*, Technique Report, The MITRE Corporation, 1990.

[9] M.Fitting, *First-order Logic and Automated Theorem Proving*, Springer-Verlag, 1990.

[10] J.H.Gallier, *Logic for Computer Science: Fundations of Automatic Theorem Proving*, Harper & Row, 1986, pp60-72.

[11] M.Gorden, *HOL: A Machine Oriented Formulation of Higher Order Logic*, Technique Report, Computer Laboratory, University of Cambridge, 1985.

[12] M.Gorden, R.Milner and C.Wadsworth, *Edinburgh LCF: A Mechanised logic of computation*, Lecture Notes in Computer Science Number 78, Springer-Verlag, 1979.

[13] R.W.Gosper, *Indefinite Hypergeometric sums in MACSYMA*, Proc. MACSYMA Users Conference, Berkeley CA, 1977, pp237-252.

[14] R.L.London and D.R.Musser, *The Application of a Symbolic Mathematical System to Program Verification*, Technique Report, USC Information Science Institute.

[15] A.Quaife, *Automated Deduction in von Neumann-Bernays-Gödel Set Theory*, Technique Report, Dept. of Mathematics, Univ. of California at Berkeley, 1989(submitted to the Journal of Automated Reasoning).

[16] E.Sacks, *Hierarchical Inequality Reasoning*, Technique Report, MIT Laboratory for Computer Science, 1987.

[17] P.Suppes and S.Takahashi, *An Interactive Calculus Theorem-prover for Continuity Properties*, Journal of Symbolic Computation, No.7, 1989, pp 573-590.

[18] E.C.Titchmarsh, *The Theory of Functions*, Oxford University Press, 1932, pp351-353.

[19] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*, Wolfram Research Inc., 1988.